

Utah State University

DigitalCommons@USU

All ECSTATIC Materials

ECSTATIC Repository

6-2020

Developing Open Source Software using Version Control Systems: An Introduction to the Git Language for Documenting Your Computational Research

Jared D. Smith

University of Virginia, js4yd@virginia.edu

Jonathan D. Herman

University of California, Davis, jdherman@ucdavis.edu

Follow this and additional works at: https://digitalcommons.usu.edu/ecstatic_all



Part of the [Civil and Environmental Engineering Commons](#), [Computer Law Commons](#), [Other Computer Sciences Commons](#), and the [Systems Engineering Commons](#)

Recommended Citation

Smith, Jared D. and Herman, Jonathan D., "Developing Open Source Software using Version Control Systems: An Introduction to the Git Language for Documenting Your Computational Research" (2020). *All ECSTATIC Materials*. Paper 87.

https://digitalcommons.usu.edu/ecstatic_all/87

This Lecture Material is brought to you for free and open access by the ECSTATIC Repository at DigitalCommons@USU. It has been accepted for inclusion in All ECSTATIC Materials by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.



Developing Open Source Software using Version Control Systems

An Introduction to the Git Language for Documenting your Computational Research

Jared D. Smith, University of Virginia
js4yd@virginia.edu

and

Jon Herman, University of California, Davis
jdherman@ucdavis.edu

Outline

1. What is Version Control? Open Source Software?
2. Why are they good for research?
3. The Git language for version control
4. How to install and use the Git language
5. Visualizing Git commands and commit history
6. Cloud-hosting Git code repositories
7. Legal Issues and Thoughts
8. Git Tutorial

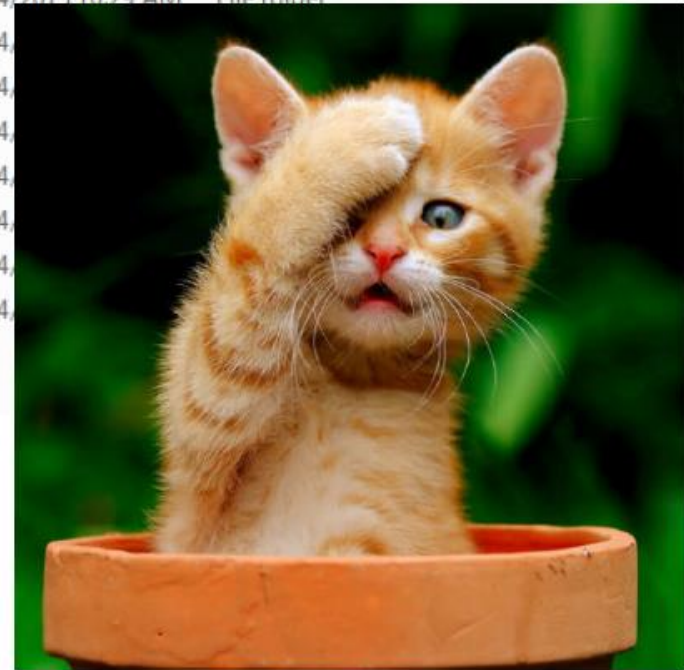
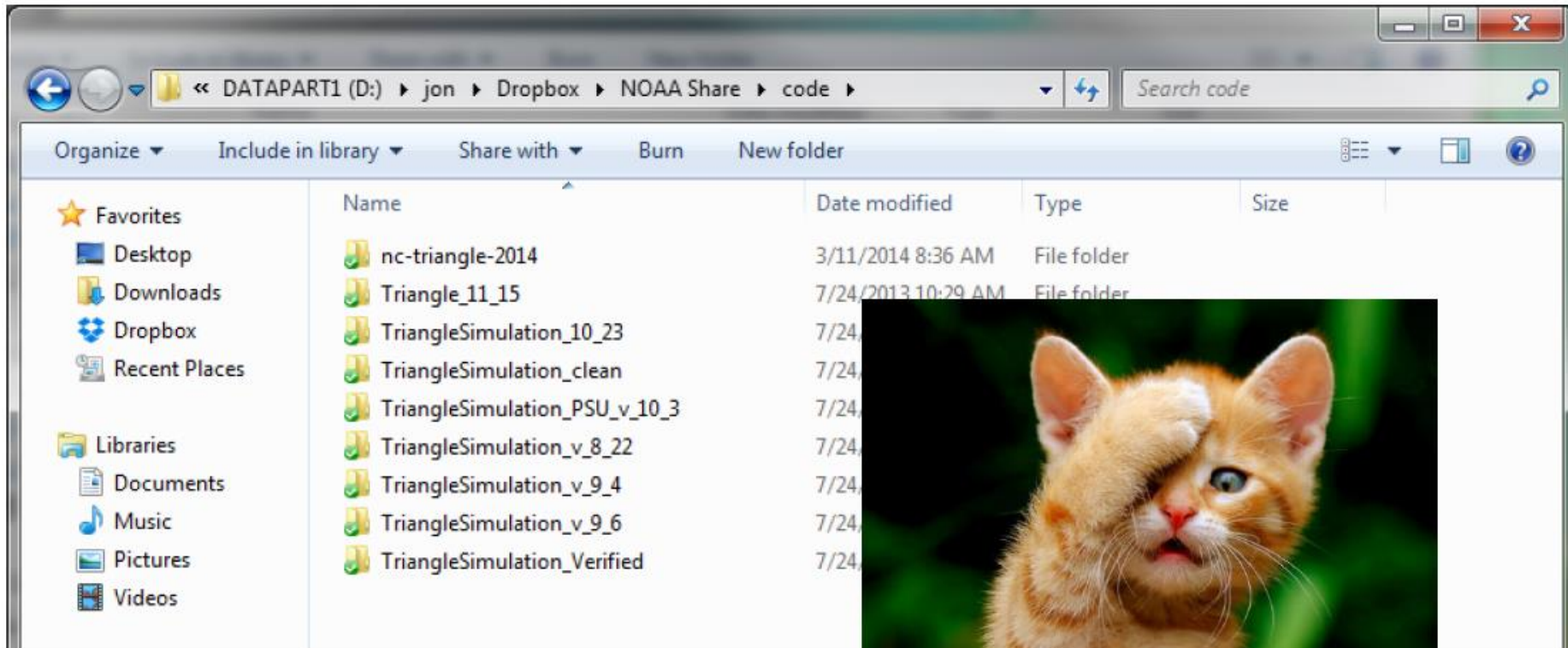
Goal: Become more familiar with using git commands

Computational research: what are the “experimental steps”?

```
1 from __future__ import division
2 import numpy as np
3 from scipy.stats import norm
4 from ..util import read_param_file
5 import common_args
6
7 # Perform Sobol Analysis on file of model results
8 # Returns a dictionary with keys 'SI', 'SI_conf', 'ST', and 'ST_conf'
9 # Where each entry is a list of size D (the number of parameters)
10 # Containing the indices in the same order as the parameter file
11 def analyze(pfile, output_file, column = 0, calc_second_order = True, num_resamples = 1000,
12            delim = ' ', conf_level = 0.95, print_to_console=False):
13
14     param_file = read_param_file(pfile)
15     Y = np.loadtxt(output_file, delimiter=delim, usecols=(column,))
16     D = param_file['num_vars']
17
18     if calc_second_order and Y.size % (2*D + 2) == 0:
19         N = int(Y.size / (2*D + 2))
20     elif not calc_second_order and Y.size % (D + 2) == 0:
21         N = int(Y.size / (D + 2))
22     else: raise RuntimeError("""
23         Incorrect number of samples in model output file.
24         Confirm that calc_second_order matches option used during sampling.""")
25
26     if conf_level < 0 or conf_level > 1: raise RuntimeError("Confidence level must be between 0-1.")
27
28     A = np.empty(N)
29     B = np.empty(N)
30     AB = np.empty((N,D))
31     BA = np.empty((N,D)) if calc_second_order else None
32     step = 2*D+2 if calc_second_order else D+2
33
```

Code!

Some familiar motivation



J. Herman (2014)



Have you ever ...

1. Tried to improve a program and broken it so badly you wished you could abandon all your changes?
2. Left a program in a broken state, and needed new results right away?
3. Tried to work on a program at the same time as others?

The “manual copy” aproach doesn’t work very well. Enter Version Control Systems (VCS) like **git** and **mercurial**.

Adapted from Lupton 2012, APC 524 course at Princeton

J. Herman (2014)

More Motivation: “The Reproducibility Crisis”

SCIENTIFIC DATA

Corrected: Author Correction

OPEN

Assessing data availability and research reproducibility in hydrology and water resources

James H. Stagge^{1,2}, David E. Rosenberg¹, Adel M. Abdallah^{1,3}, Hadia Akbar¹, Nour A. Attallah¹ & Ryan James¹

Received: 22 October 2018

Accepted: 23 January 2019

Published: 26 February 2019

There is broad interest to improve the reproducibility of published research. We developed a survey tool to assess the availability of digital research artifacts published alongside peer-reviewed journal articles (e.g. data, models, code, directions for use) and reproducibility of article results. We used the tool to assess 360 of the 1,989 articles published by six hydrology and water resources journals in 2017. Like studies from other fields, we reproduced results for only a small fraction of articles (1.6% of tested articles) using their available artifacts. We estimated, with 95% confidence, that results might be reproduced for only 0.6% to 6.8% of all 1,989 articles. Unlike prior studies, the survey tool identified key bottlenecks to making work more reproducible. Bottlenecks include: only some digital artifacts available (44% of articles), no directions (89%), or all artifacts available but results not reproducible (5%). The tool (or extensions) can help authors, journals, funders, and institutions to self-assess manuscripts, provide feedback to improve reproducibility, and recognize and reward reproducible articles as examples for others.

More Motivation: “The Reproducibility Crisis”

Many published studies are difficult or impossible to reproduce, given the paper and any supplied documentation alone.

Others require significant technical support from experts who have developed the code.

Only the code that ran the experiments can be used to exactly reproduce them.

Modelers are biased by their education and experiences, and each programmer will make their own assumptions, which may or may not be documented in academic publications or reports.

Even code that is provided and reproducible may be impossible to understand – the ultimate goal of reproducibility is on out-of-sample tests to develop consensus (replicability)

What is the main contribution of computational research? The paper? The code?

“The idea is: An article about computational science in a scientific publication is *not* the scholarship itself, it is merely *advertising* of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.” David Donoho, 1998.



Publish your computer code: it is good enough

Freely provided working code — whatever its quality — improves programs and enables others to engage with your research, says Nick Barnes.

**NOBODY IS ENTITLED
TO DEMAND
TECHNICAL SUPPORT
FOR FREELY
PROVIDED CODE:
IF THE FEEDBACK
IS UNHELPFUL,
IGNORE IT.**

“Code isn’t very polished”

→ Performs algorithm described in paper? If yes, publish; if no, fix.

“It’s not common practice”

→ This should change

“It’s the IP of my institution”

→ Sometimes, if new discovery. Pay attention to licensing.

“People will criticize and demand technical support” (see left)

Version Control

Keeping track of all code changes across developers



Open Source

Sharing your code online



THE FUTURE

(is now!)

Why are version control and open source software good for research?

- Always know where to find the latest version of a project
- Simple to share - no passing of zip folders by email
- Easy to modify without breaking the code
- Easy to work with collaborators from other institutions
- Good way to document the exact code used to reproduce studies – with complete revision history.
 - Easy to change in the future, if needed
- Promotes your work
- Saves time
- Cloud backup for your code

What is git?

- An open source distributed version control system
- Developed by Linus Torvalds in 2005 to keep track of Linux kernel development (“distributed” is important here)
- Now the most widely used VCS
- Originally a command line tool, but desktop versions are available



J. Herman (2014)

Useful Resources to Learn the Git Language

AAWATS (Ashley Watson?)
[dictionary](#)

Jon Herman's Blog Posts
([Intro to Git](#), [Working with Remotes](#))

[Tutorials from Atlassian](#)
on All Git Commands

Rarely need more than the
commands on the cheat sheet

[Some visualizations of commands](#)



GitHub
GIT CHEAT SHEET
v1.11.1

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT
GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
<https://windows.github.com>

GitHub for Mac
<https://mac.github.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
<http://git-scm.com>

CONFIGURE TOOLING
Configure user information for all local repositories

<code>\$ git config --global user.name "[name]"</code>
Sets the name you want attached to your commit transactions
<code>\$ git config --global user.email "[email address]"</code>
Sets the email you want attached to your commit transactions
<code>\$ git config --global color.ui auto</code>
Enables helpful colorization of command line output

CREATE REPOSITORIES
Start a new repository or obtain one from an existing URL

<code>\$ git init [project-name]</code>
Creates a new local repository with the specified name
<code>\$ git clone [url]</code>
Downloads a project and its entire version history

MAKE CHANGES
Review edits and craft a commit transaction

<code>\$ git status</code>
Lists all new or modified files to be committed
<code>\$ git diff</code>
Shows file differences not yet staged
<code>\$ git add [file]</code>
Snapshots the file in preparation for versioning
<code>\$ git diff --staged</code>
Shows file differences between staging and the last file version
<code>\$ git reset [file]</code>
Unstages the file, but preserve its contents
<code>\$ git commit -m "[descriptive message]"</code>
Records file snapshots permanently in version history

GROUP CHANGES
Name a series of commits and combine completed efforts

<code>\$ git branch</code>
Lists all local branches in the current repository
<code>\$ git branch [branch-name]</code>
Creates a new branch
<code>\$ git checkout [branch-name]</code>
Switches to the specified branch and updates the working directory
<code>\$ git merge [branch]</code>
Combines the specified branch's history into the current branch
<code>\$ git branch -d [branch-name]</code>
Deletes the specified branch

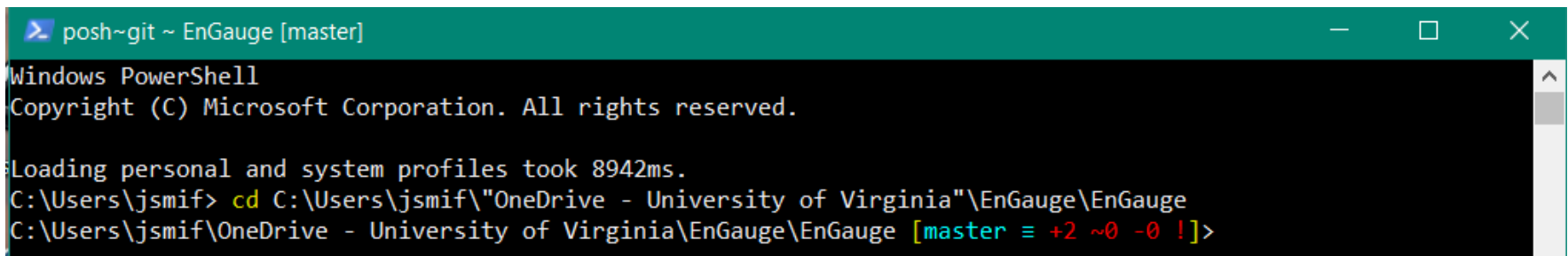
[Cheat Sheet](#)

How to install git

- On Debian/Ubuntu: `sudo apt-get install git`
- On Mac/Windows: <http://git-scm.com/downloads>
- Also available as a Cygwin package
- Available on most clusters (run “module load git”)

Additional add-ins:

- Windows PowerShell with Git is nice



```
posh~git ~ EnGauge [master]
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Loading personal and system profiles took 8942ms.
C:\Users\jsmif> cd C:\Users\jsmif\OneDrive - University of Virginia\EnGauge\EnGauge
C:\Users\jsmif\OneDrive - University of Virginia\EnGauge\EnGauge [master ≡ +2 ~0 -0 !]>
```

Modified from J. Herman (2013)

Git Commands

git commit: save the current state

Without git



A change



...

With git



Each commit (checkpoint) saves a message, the date/time, and ID of the person saving it.

The “current state” refers to all files in the repository.
You can choose which files to include/exclude.

Modified from J. Herman (2014)



git log: view entire revision history

```
jdherman /cygdrive/d/jon/code/pdfextract (master)  
$ git log
```

```
commit dca3c012e391741d4c5c4e921b266af978815ac9  
Author: Karl Jonathan Ward <karl.j.ward@gmail.com>  
Date: Mon Jul 21 15:52:25 2014 +0100
```

```
extract-bib example in readme
```

```
commit 358df84f6c434f9c6275f4caac5dcc0ff044e177  
Merge: 4ade684 3358398  
Author: Karl Jonathan Ward <karl.j.ward@gmail.com>  
Date: Mon Jul 21 15:34:08 2014 +0100
```

```
Merge pull request #11 from jdherman/feature/bibtex-resolve
```

```
Fix reference resolver, and add option to output BibTeX with the CrossRef API
```

```
commit 3358398dfe6efb52e826b7d89bb08c6eaf4dc093  
Author: jdherman <jdherman8@gmail.com>  
Date: Mon Jul 21 10:20:35 2014 -0400
```

```
add v1 to api link
```

Commit ID

Who did it

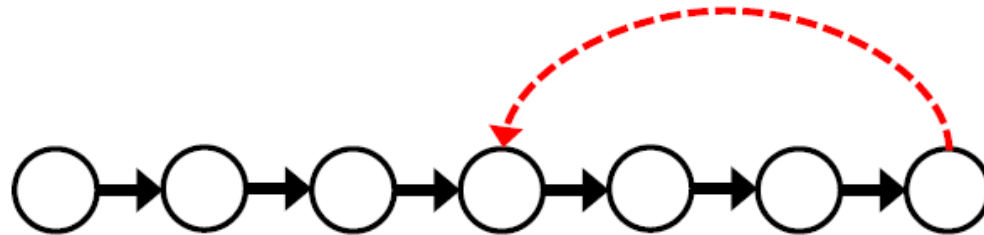
When they did it

What they did



J. Herman (2014)

`git reset`: **reset** to a prior state



The undone commits can either be kept (a “soft reset”) or removed (a “hard reset” – careful with this).

Example: `git reset --hard HEAD~1`

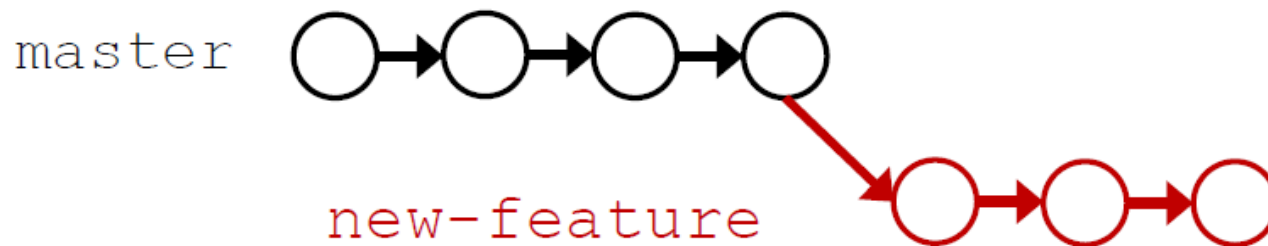
Reset back 1 commit. Can change 1 to any number of commits.

[Git undo resource](#)

Modified from J. Herman (2014)



git branch: create a new branch



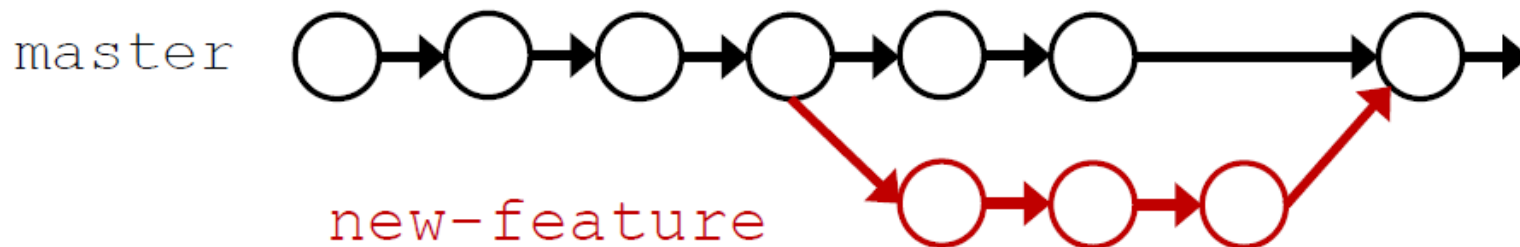
All commits done in the new branch do not affect master
This is incredibly helpful for testing large changes/additions

Be aware that if you switch branches, your directory will change the files it has in it.

Modified from J. Herman (2014)



git merge: add changes from other branch



Or, if the new feature doesn't pan out, just go back to `master` and delete `new-feature`

Merge conflicts will only occur if the same code was edited in both branches. Resolve manually.



Pro Tips for Git

Entering ‘git’ will show a list of common commands

‘git status’ displays all modified and untracked files

Enter ‘q’ to escape from a long list of messages

When your repository has multiple branches, it can be useful to have the branches in separate directories

Only keep source code under version control. You can have other stuff in the folder (executables, data, etc.) but just don’t add it to be tracked by git. – J. Herman

[gitignore](#) files are great for this!

Free Cloud Services/Hosts with Git Support

GitHub

GitLab - might be better for hosting private repositories

BitBucket – by Atlassian

And [many others](#)

Important: git and GitHub are not the same thing

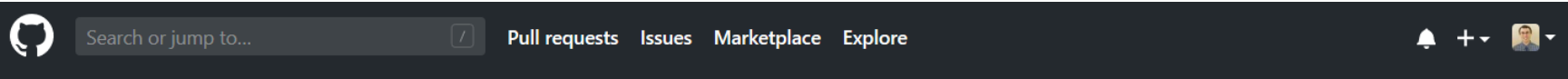
- **git** is an open-source distributed version control system (<http://git-scm.com/>)
- **GitHub** is a company that provides hosting for git repositories (<https://github.com/>)

Ok, so what about GitHub?

How could you share a code repository with many people?

- GitHub and other cloud services provide hosting, bug tracking, and other tools to help you make sense of your revision history.
- Any repositories that you're willing to open source are **free** to host on GitHub.
- If you want private repositories, it costs (not much) money, and is free for students and academics with a *.edu email address.

GitHub – LinkedIn for Code



😊 Set status

Jared D Smith

jds485

★ PRO

Edit profile

👤 University of Virginia
📍 Charlottesville, VA
✉ jds485@cornell.edu
🌐 <https://www.linkedin.com/pub/jared-...>

Overview Repositories 5 Projects 0 Packages 0 Stars 3 Followers 6 Following 22

Pinned

Customize your pins

📄 calvinwheaton/geothermal_pfa

Code related to geothermal research

● R ★ 1

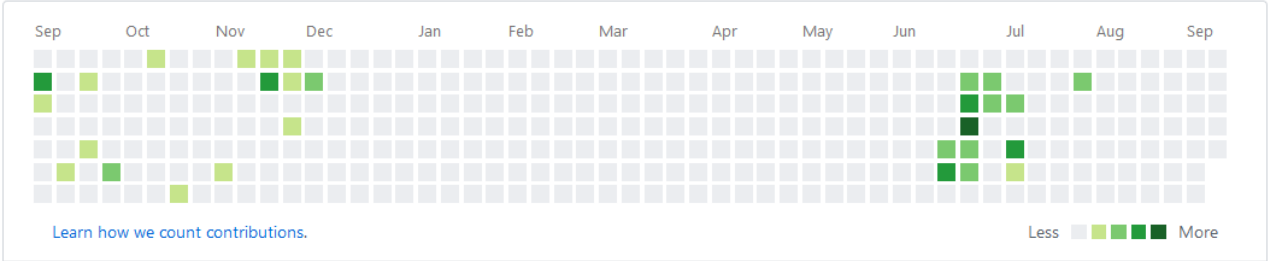
📄 Geothermal_DataAnalysis_CrossSections

This repository contains code that may be useful for processing geothermal bottomhole temperature (BHT) data into a useful format for spatial outlier analyses and spatial regressions. Plot template...

● R ★ 1

103 contributions in the last year

Contribution settings ▾



Contribution activity

2019

August - September 2019

2018



What's included in a GitHub Repository?

Sensitivity Analysis Library in Python (Numpy). Contains Sobol, Morris, and FAST methods.

<http://jdherman.github.io/SALib/> — Edit

84 commits

Full revision history

0 releases

2 contributors

branch: master

scaling and CV seem to work

jdherman authored 21 days ago

latest commit 1a7316b5f5

SALib	scaling and CV seem to work	21 days ago
examples		21 days ago
.gitignore		23 days ago
LICENSE.md		a year ago
README.md	readme	23 days ago

Browse files online

Your Code

README.md

Documentation!

Sensitivity Analysis Library (SALib)

Python implementations of commonly used sensitivity analysis methods. Useful in systems modeling to calculate the effects of model inputs or exogenous factors on outputs of interest.

Requirements: [NumPy](#), [SciPy](#)

Methods included:

J. Herman (2014)

Code

Issues 1

Pull Requests 0

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:jdherman

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).

Clone in Desktop

Download ZIP

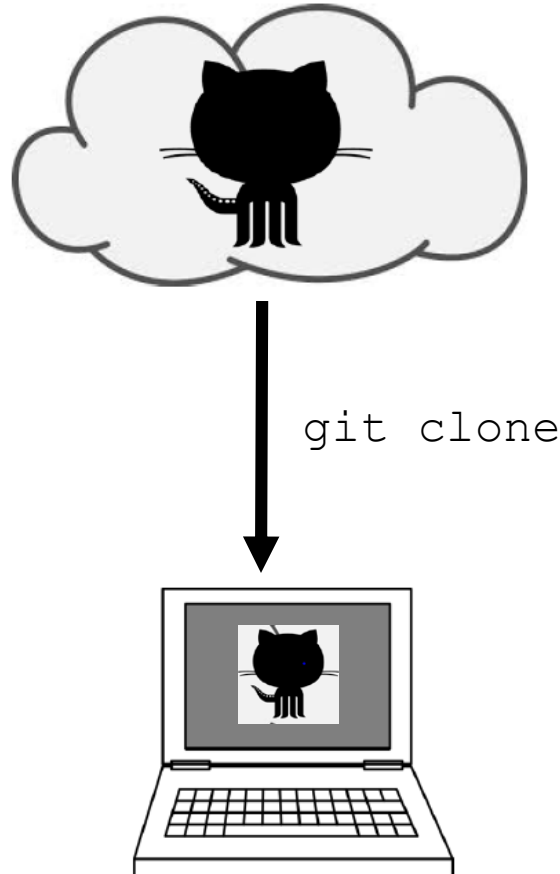
Download

Detailed Contents of a Code Repository

1. README.md file. The .md is for the Markdown language. It's simple to use, and there are [many tutorials](#).
2. Your well-commented code
3. Test functions for your code (optional, but I think they should be required upon release)
4. Example data – may be better to host large files elsewhere
5. License file. GitHub and other services have built-in standard licenses
6. .gitignore file. This file lists all of the files (or extensions) to be excluded from git tracking. GitHub has pre-made files for all code languages, but you have to select them.

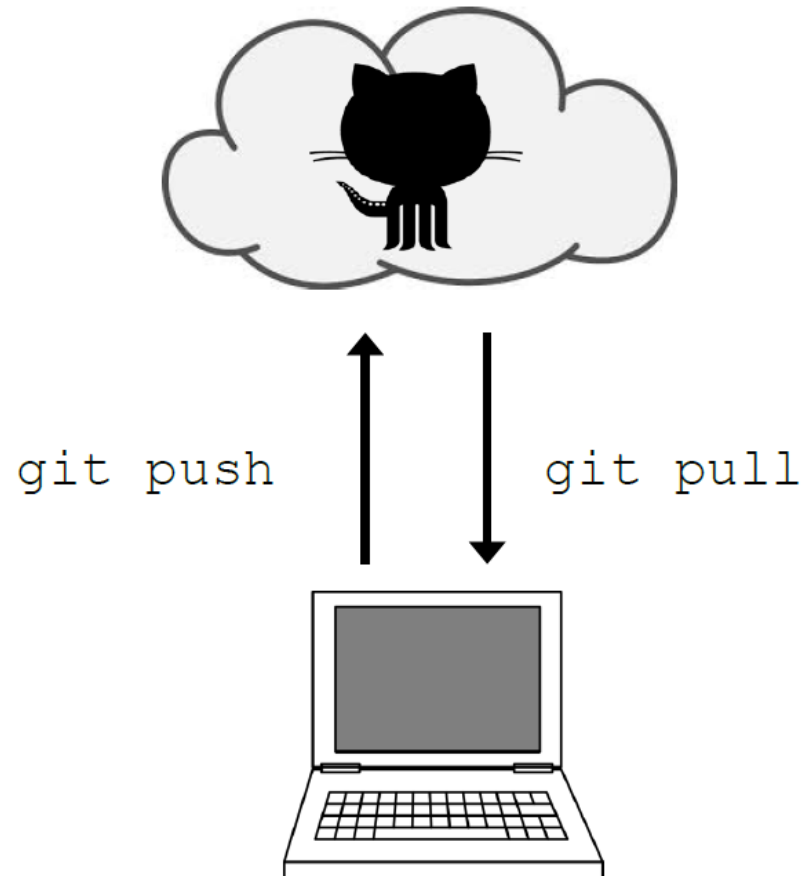
Getting Code from GitHub

`git clone`: **Gets (clones) code from GitHub to your desktop**



Committing and Gathering Code Changes using GitHub

`git push/pull`: when you own the repository



Sourcetree GUI – Visualizing your Commits

The screenshot displays the Sourcetree GUI interface. At the top, there are menu options: File, Edit, View, Repository, Actions, Tools, Help. Below the menu is a toolbar with icons for Commit, Push, Pull, Fetch, Branch, Merge, Stash, Discard, and Tag. The main window shows a list of repositories: jaredthermalcal..., Geothermal_D..., PorosityPerme..., geothermal_pfa, Geothermal_E..., HeatInPlaceCa..., FractureOrient..., and GEOPHIRES. The central pane shows a commit history table with columns for Graph, Description, Date, Author, and Commit. A callout box labeled "All of your repositories" points to the repository list. Another callout box labeled "Your Commit History" points to the commit history table. The table lists several commits, with the most recent one highlighted: "origin/calvinSA Detecting number of COSUNA variables automatically for each column" on 4 Feb 2019 12:58 by Jared D Smith. Below the table, the commit details for the selected commit are shown, including the commit hash, author, date, and committer. A callout box labeled "What changed since last commit" points to the file status section, which shows "MainScriptSA.py" as the changed file. The bottom pane shows the code diff for "MainScriptSA.py", with a callout box pointing to the changes. The bottom of the window has tabs for File Status, Log / History, and Search.

Graph	Description	Date	Author	Commit
	Uncommitted changes	12 Sep 2019 14:21	*	*
o	origin/calvinSA Detecting number of COSUNA variables automatically for each column	4 Feb 2019 12:58	Jared D Smith <jds	e541261
	Edited comments to fix a typo.	23 Dec 2018 15:49	Jared D Smith <jds	bac0deb
	Added function used for parallel computation by individual wells/records	23 Dec 2018 15:36	Jared D Smith <jds	ea23889
	Added beta and lognormal distributions. Edited and added comments	23 Dec 2018 15:35	Jared D Smith <jds	47f2e89
o	master origin/master origin/HEAD Update	23 Dec 2018 12:43	Jared D Smith <jds	323425f
	Edited comments, added white space	23 Dec 2018 12:36	Jared D Smith <jds	ef87235
	Added white space and a fixme idea for more exact interpolation	23 Dec 2018 11:28	Jared D Smith <jds	20a5970
	Edited comments, added white space, added a fixme idea	23 Dec 2018 11:21	Jared D Smith <jds	c8c0676
	Edited comments.	23 Dec 2018 11:13	Jared D Smith <jds	ea7582f
	Deleted the No BHT Correction functions because one can use	23 Dec 2018 11:04	Jared D Smith <jds	79b8e9
	Edited comments and added white space between some oper	22 Dec 2018 11:43	Jared D Smith <jds	82ea3ff
	Edited comments	22 Dec 2018 11:27	Jared D Smith <jds	ba45557
	Removed some outdated FIXMEs and edited comments.	22 Dec 2018 11:11	Jared D Smith <jds	fc6a2d0
	Converting strata thicknesses and conductivities to float64 values when they are read in using CsvF	22 Dec 2018 10:57	Jared D Smith <jds	e3aa713

Commit: e5412616d563bd852ddc219cc5b364af147417fa [e541261]
Parents: bac0deb583
Author: Jared D Smith <jds485@cornell.edu>
Date: Monday, February 4, 2019 12:58:22 PM
Committer: Jared D Smith

Detecting number of COSUNA variables automatically for each column

```
MainScriptSA.py
51 + #Fixme: add variable to indicate if user wants detailed or
52 +
53 + %% functions from files
54 + from CsvReader import read_strats
55 + from ThermModLoopGenInputs import calcSaveSA
56 + from DeStratCOSUNAWriteProbDistSA import makeDetailStratPro
57 + from UniformMakeWrite import makeUniformSamples
```

```
'seds_baseament_km_db1bnd': np.
'seds_map_km': np.float64,
'surf temp c': np.float64,
'long': np.float64,
'lat': np.float64,
'elev_m': np.float64,
'base_cond': np.float64,
'base_cond_sd': np.float64,
```

Sourcetree GUI – Visualizing your Commits

Can track all of your repositories in one place

Shows all of your branches, and how many commits ahead branches are from each other

Shows the difference between current changes and previous commits for all code that has changed

Can be used instead of command line

- Sometimes it's better, especially for merging branches. The GUI has nice a side-by-side comparison of conflicts.

Legal: Licensing Code

GPL

GPL is the most widely used free software license and has a strong copyleft requirement. When distributing derived works, the source code of the work must be made available under the same license. There are multiple variants of the GPL, each with different requirements.

GNU GPL v3.0 GNU GPL v2.0 GNU Affero GPL v3.0

Required	Permitted	Forbidden
<ul style="list-style-type: none">● Disclose Source● License and copyright notice● State Changes	<ul style="list-style-type: none">● Commercial Use● Distribution● Modification● Patent Grant● Private Use	<ul style="list-style-type: none">● Hold Liabile● Sublicensing

[View full GNU GPL v3.0 license »](#)

MIT

A permissive license that is short and to the point. It lets people do anything with your code with proper attribution and without warranty.

Required	Permitted	Forbidden
<ul style="list-style-type: none">● License and copyright notice	<ul style="list-style-type: none">● Commercial Use● Distribution● Modification● Private Use● Sublicensing	<ul style="list-style-type: none">● Hold Liabile

[View full MIT License license »](#)

<http://choosealicense.com/licenses/>

Legal: Licensing Code

Default Licenses are available on GitHub, BitBucket, etc.

These licenses may be good as-is, but it may be in your best interest to modify them

Working with a lawyer may be required for larger projects that become commercialized.

Many repositories have clauses that you must cite them if you use their code in your work. Be careful, and cite everything you use to avoid legal complications.

Git Tutorial #1: Clone Laurence Lin's RHESSysEastCoast Ecohydrological Model GitHub Repository

1. Create a directory on your computer where you want to keep the repository
2. Navigate to that directory using the command line terminal of your choice (this can also be completed using a GUI of your choice [maybe not on Linux?])
3. For Linux, load the git module
4. Clone the [repository](#) into the directory, and change into the code directory
5. List the branches of that repository (use -a to see all branches)
6. Make your own branch in that repository
7. Check out your branch
8. Look at previous commits, and find the commit from Sept 13, 2019
9. Reset the HEAD back to that commit
This is extremely useful to know how to do (mistakes in coding happen)
10. Check the status of the contents in the repository
11. Do a diff to see what changed
12. Add and commit the changed files to your branch (with a commit message)
13. Check the status again
14. List files in the repository

Tutorial Extras – adding and changing your files

1. Manually add a new .csv file on your branch and save it
2. Manually add a comment to a different file (e.g. README) in your branch and save it
3. Check the status of the contents in the repository
4. Pretend you don't want to track files with a .csv extension. Let's [gitignore](#) them!
5. Check the status again
6. Add the files to be staged for commit (tab complete helps)
7. Commit the files to your branch (with a commit message)
8. Check the status again
9. Undo that commit with a reset of the HEAD
10. (optional) Delete your branch

Git with Sourcetree Tutorial Videos

Tutorial 1: Introduction to git Commands Using Sourcetree

Tutorial 2: More Advanced git Commands with Sourcetree

- multiple branches
- working with remote repositories